Directed Taskgraph Scheduling Using Simulated Annealing

Erik H. D'Hollander and Yves Devis
Department of Electrical Engineering
State University of Ghent
B-9000 Ghent, Belgium

Abstract

Simulated annealing is recognized as a novel method to optimize the load in multicomputer systems, subject to the interprocessor communication overhead. Recently, highly nonlinear mapping and load balancing of undirected taskgraphs has been solved in a successful way. In this paper the scope is extended to directed taskgraphs, representing the data and control dependencies in common programs. The annealing algorithm operates in stages. In each stage an annealing packet of ready tasks is formed and the tasks are allocated to the idle processors. The cost function is based on the priority level of the tasks in the taskgraph and the intertask communication requirements. The resulting schedule of four programs on three architectures show a significant speedup improvement compared to the Highest Level First list algorithm.

1 Introduction

For the execution on a multiprocessor a program is partitioned into tasks and these tasks are allocated to the available processors. The scheduling process must pursue two conflicting objectives: to maximize the processor utilization and to minimize the inter processor communication. This problem is known to be NP-complete and a solution is approximated by suboptimal heuristics such as the well known Highest Level First (HLF) list algorithm [7,1,9]. These algorithms have proven adequate when the communication overhead is moderate, e.g. for strongly coupled shared memory multiprocessor systems.

Since the revival of neural networks several assignment problems have been addressed by simulated annealing. In these problems an undirected taskgraph is mapped onto a machine graph. An undirected taskgraph represent a set of communicating tasks without precedence constraints. The cost function aims to balance the load subject to a minimal communication overhead.

In this paper a directed graph is scheduled by simulated annealing. The algorithm performs a load balance and minimizes the communication, subject to the precedence constraints between the tasks.

[1]This research was supported by the Belgian Ministery of Science, under the contract OOA-87/93-117.

In particular the algorithm takes into account the changing communication patterns during the execution of taskgraph. The performance of simulated annealing scheduling has been measured by simulating the execution of four programs on three different multicomputer topologies. In all cases simulated annealing outperformed the best list-algorithm.

2 Definitions and Notations

Host Configuration

Consider a distributed processing system $HC = \{P, L\}$ consisting of a set of processors P and an interconnection network L. The N_P processors are represented by $P = \{p_i, i = 1, ..., N_P\}$. The network topology is described by the *processor interconnection matrix* L, where $l_{ij} = 1$ indicates the presence of a point-to-point link between two processors p_i and p_j. This includes a bus (star), a hypercube or a ring network. The *distance* $d(i,j)$ between two processors equals the number of links on the shortest path joining the processors p_i and p_j. The links are bidirectional (L is symmetrical), have a bandwidth BW (Mbits per second) and can carry only one message at a time. It is assumed that incoming messages preempt an active processor.

Taskgraph

The program is partitioned into a directed taskgraph $TG = \{T, R, W, <*\}$. This quadruple consists of the set of tasks $T = t_i, i = 1, ..., N_T$, the load requirements $R = \{r_i\}$, the communication weights $W = \{w_{ij}\}$ and the precedence constraints $<*$. The nodes t_i represent tasks and have an estimated CPU-load r_i. The edges are labeled with weights w_{ij}, indicating the communication time between task t_i and task t_j. $t_i <* t_j$ indicates that t_j must start after the termination of t_i. t_i is a *predecessor* of t_j and t_j is a *successor* of t_i.

Simulated Annealing

With the arrival of neural networks, statistical methods have gained success in the area of highly complex and combinatorial optimization problems with many interacting variables [10]. Most of these problems are NP-complete, and require ingenious heuristic approaches. Yet often the heuristics are trapped in local minima of the multidimensional cost surface. Simulated annealing is able to overcome this barrier by statistical hill climbing. Instead of

following a steepest descent trajectory, the path is perturbed by random walks with a decreasing probability. The minimization process is controlled by a cooling temperature which makes the trajectory evolve from a purely random walk towards a deterministic path. The idea is to find the global minimum by escaping the local cavities during the cooling process.

The simulated annealing technique is governed by the following components: the *mapping function*, the *cost function*, the *mapping scheme*, and the *cooling function*.

The *mapping function* $m : T \rightarrow P$ assigns the tasks to the processors, such that $p_k = m(t_j)$ if task t_j is allocated onto processor p_k. The *cost function* $F(m)$ measures the quality of the mapping with respect to the load balance and the communication overhead.

The *mapping scheme* randomly redistributes the allocation of tasks to processors, thereby producing a new mapping function m'. The simulated annealing process will accept the new mapping m' depending on the cost $F(m')$ and the temperature $Temp$, with a probability

$$B(F, Temp) = \frac{1}{1 + e^{-\frac{F}{Temp}}}. \quad (1)$$

where $0 \leq Temp \leq \infty$. For extreme values $Temp = 0$ and $Temp = \infty$, the mapping m' is accepted with the following probabilities:

$$\begin{aligned} B(F, \infty) &= .5 \\ B(F, 0) &= \begin{cases} 1 & \text{if } F < 0 \text{ (accept move)} \\ 0 & \text{if } F \geq 0 \text{ (reject move)} \end{cases} \end{aligned} \quad (2)$$

The *cooling function* generates a sequence of temperatures $Temp_i$, varying from ∞ (an arbitrary acceptance) to 0 (a deterministic acceptance). The cooling policy influences the convergence speed and the quality of the obtained solution.

3 Related Work

The assignment problems solved using simulated annealing differ by the assumptions on the host architecture, the taskgraph and the cost function. Depending on the number of tasks and processors, the following assignment schemes were investigated:

- The mapping problem [3]: $N_T \leq N_P$, $<* = \emptyset$.
- The balancing problem [8]: $N_T > N_P$, $<* = \emptyset$.
- The scheduling problem (this paper): $N_T > N_P$, $<* \neq \emptyset$.

In the *mapping problem* [3], Bollinger and Midkiff map an undirected taskgraph on the host architecture. There is at most one task per processor and the objective is to minimize simultaneously the total communication and the maximal point-to-point communication on a single link. The authors take into account the communication weight between the tasks, W and allow arbitrary routing. The simulating annealing approach allowed to adopt a more realistic communication model than the one used in other approaches [2,11], except for the condition $N_T < N_P$.

Hwang and Lee removed the restriction on the number of tasks in the *balancing problem* [8]. With more tasks than processors there are two objectives: to balance the load and to minimize the interprocessor communication. The cost function therefore has a balance term and a communication term. The balance term sums the absolute deviation from the average processor load and the communication term sums the traffic on the interprocessor links. In the balancing problem it is assumed that all modules execute concurrently and communicate during the whole execution of the program. While this is true when the modules are independent (e.g. production systems), in many partitioned programs data and control dependencies create precedence constraints. In this case one has a *scheduling problem*. A load balancing scheme which takes into account the precedence rules to solve the scheduling problem is presented in this paper.

4 The Scheduling Problem

4.1 Annealing Packets

In programs characterized by a directed taskgraph, the communication and the load patterns vary largely during the execution time, invalidating the assumptions of the balancing problem. We solve the scheduling problem by creating *annealing packets* at discrete assignment epochs. The first epoch is at time zero and successive epochs occur when one or more processors become idle. An annealing packet contains the ready tasks and the idle processors. The ready tasks have no unfinished predecessors. At each epoch a simulated annealing process maps the tasks of one packet onto the processors. Unassigned tasks are moved to the following annealing packet and new annealing packets are created until all tasks are assigned. The tasks compete for an assignment based on their priority and on the communication overhead with the other tasks.

4.2 Cost Function

The cost function consists of a load balancing term and a communication term.

a) Load Balancing Cost

The *critical path* of a directed taskgraph consists of the longest chain joining the root task and a leaf task. In

order to minimize the execution time, the cost function must encourage the assignment of tasks on the critical path. Therefore tasks are given a priority measured by the *tasklevel* [4]. The level n_i of a task t_i equals the accumulated execution time of every task on the longest path connecting t_i with a leaf task. In other words, in a system with an arbitrary number of processors and no communication overhead, the tasklevel represents the minimal remaining execution time when the task is started. The annealing process should favor the selection of high-level tasks. This is realized using the following load balancing cost function

$$F_b = -\sum_{i=1}^{N} n_i s(i) \tag{3}$$

N is the number of task in the annealing packet, $s(i) = 1$ when task t_i is selected, else $s(i) = 0$. Minimizing this function corresponds with assigning first the highest level tasks of the annealing packet.

b) Interprocessor Communication Cost

Two parameters characterize the cost of sending a message between processors p_x and p_y: σ, the time to forward one message and τ, the time to receive or to route one message. These parameters account for the following events: the context switches (S) to save and restore the processor state, the output setup (O) to prepare the I/O hardware and the header control (H) to determine if an incoming message needs to be routed to other processors. With these parameters, one has

$$\begin{aligned} \sigma &= 2S + O \\ \tau &= 2S + H + O \end{aligned}$$

For the bit-serial linked hypercube processor systems the parameters were set to $O = 3\mu s$, $S = H = 2\mu s$; this gives $\sigma = 7\mu s$ and $\tau = 9\mu s$.

On a connection link of BW bits per second, the time to carry a message of length L over a path between processors i and j equals

$$w_{ij} = \frac{L}{BW}$$

The effective communication cost c_{ij} to send a message of weight w_{ij} between tasks t_i and t_j located at processors $m(t_i)$ and $m(t_j)$ respectively is

$$c_{ij} = w_{ij} d_{ij} + (d_{ij} - 1 + \delta_{m_i m_j})\tau + (1 - \delta_{m_i m_j})\sigma \tag{4}$$

where δ_{ij} is the Kronecker delta. The communication cost has three parts.

1. The distance-volume product measures the communication time on the links connecting the two processors $m(t_i)$ and $m(t_j)$.
2. The intermediate processors contribute by routing the message. This term vanishes if t_i and t_j are located on neighboring processors.
3. The third term represents the extra cost to setup a communication link. This term vanishes if both tasks reside on the same processor.

The communication cost of the annealing packet is defined

$$F_c = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N_T} c_{ij} \tag{5}$$

c) Normalized cost function

For different architecture graphs and taskgraphs, the load and communication terms can vary widely. A simple addition of the load balancing and the communication terms could outweigh one cost and discard the other. Therefore the load balancing and the communication costs are normalized, each by their proper range. The range of the balancing term is

$$\Delta F_b = (Max - Min)/N_{idle}$$

where Max and Min represent the cumulative level values when the N_{idle} free processors would execute the tasks with the highest or the lowest levels respectively. The communication range is obtained by placing the tasks with the highest communication at the largest distance, giving an estimate of the maximum communication cost, ΔF_c.

The cost function is a weighted sum of the normalized communication and load balancing terms,

$$F(m) = w_c \frac{F_c}{\Delta F_c} + w_b \frac{F_b}{\Delta F_b} \tag{6}$$

This function minimizes the communication and balances the load, while the weight factors w_b and w_c allow to emphasize one or the other element in the cost function. They are choosen such that $w_b + w_c = 1$ and can be tuned to optimize the allocation for the highest speed-up.

5 Annealing Algorithm

For notational convenience, we introduce the following abbreviation for the mapping function:$m_i = m(t_i)$.

Until all tasks $t_i \in T$ are assigned, do:

1. Assemble an annealing packet (AP) consisting of the free processors and the ready tasks (i.e. task without unfinished predecessors).
2. **for** cooling temperatures $Temp_k$, $k = 1, ..., N_I$ until convergence or until exceeding the maximum number of iterations, N_I, **do**:

(a) Arbitrarily select a task t_i and a processor p_j, where $p_j \neq m_i$.

- If processor p_j is idle, assign t_i to p_j (possibly by removing t_i from another processor): $m_i := p_j$;
- If processor p_j is busy executing $t_j \in AP$, exchange t_i and t_j: $m_i := p_j$, $m_j := p_i$.

(b) Accept the assignment with a probability given by the Boltzmann function $B(F, Temp_k)$ (equation 1).

endfor

3. Repeat from (1) if not all tasks are assigned.

6 Experimental Results

Four programs were scheduled on three different multicomputer architectures. In each case the execution was simulated to record the achieved speedup. The performance analysis [5] covers both the simulated annealing process as the speedup improvement over scheduling by the HLF-list algorithm.
The scheduled programs are:

1. Newton-Euler Inverse Dynamics for robot control (NE)
2. Gauss-Jordan linear system solver (GJ)
3. Matrix multiply (MM)
4. Fast Fourier Transform (FFT)

The programs GJ, FFT and MM are partitioned into vector operations and the NE program consists of scalar operations. The taskgraph characteristics are given in Table 1. The communication time is calculated for a 10Mb/s link between two processors and 40 bit data per variable. Extra communication overhead occurs due to the cost to send, route and receive the messages (equation 4).
The taskgraphs were mapped onto the following architectures:

1. A Hypercube with 8 processors
2. A Bus (star) topology with 8 processors
3. A Ring topology with 9 processors

a) Annealing Process Figure 1 shows the trajectories of the level-, the communication- and the total cost, F_b,F_c, F_{tot} (equations 3, 5, 6) of one annealing packet in the Newton-Euler problem. It can be seen that the annealing process decreases both the balancing and the communication costs. The program contains 95 tasks, which are assigned in 65 annealing packets. On the average there are 15 candidates for 1.46 free processors. The annealing stops when the cost function remains constant for five iterations, or when a preset maximum number is reached.
b) Speedup To estimate the speedup improvement over an heuristic task placement by the Highest Level First (HLF) algorithm, a simulation program was developed which accurately records the execution and interprocessor communication. Figure 2 shows the start of the Newton-Euler program partitioned on an 8 processor hypercube.
Furthermore Table 2 gives the speedups for both the simulated annealing and the heuristic HLF algorithm.
These results give rise to two observations. First, when the communication is not taken into account, simulated annealing gives the same or slightly better results than the HLF algorithm. This occurs despite the fact that an extensive statistical comparison of various list algorithms indicates that the HLF generated schedules remain within 5% of the optimal solution in all but one of 900 random generated taskgraphs [1]. Moreover we observed that the SA algorithm is able to optimally solve the Graham list scheduling anomalies [6]. Second, the simulated annealing algorithm outperforms the HLF algorithm by 3.5 to 52 %. This reveals that this altorithm is a worthwile alternative to the arbitrary placement of the HLF-tasks, when the interprocessor communication is not neglectable.

7 Conclusion

In recent years, simulated annealing has been recognized as a novel method to balance the load in loosely coupled multicomputers. We extended the use of simulated annealing to the scheduling of directed taskgraphs. This implies minimizing the communication and balancing the processor load, while preserving the data and control dependent precedence constraints. The results indicate that the presented algorithm is able to improve the speedup in real program taskgraphs by more than 50%.

References

[1] Adam T.L., Chandy K.M., Dickinson J.R., *A comparison of list schedules for parallel processing systems,* Communications of the ACM 17, 12, 685-690, 1974

[2] Bianchini R.P., Shen J.P., *Interprocessor Traffic Scheduling Algorithm for Multiple-Processor Networks,* IEEE Trans. on Computers 36, 4, 396-409, 1987, Vol. 36, 4, pp. 396-409, 1987

[3] Bollinger S. Wayne, Midkiff Scott F., *Processor and Link Assignment in Multicomputers using Simulated*

Annealing, Proceedings of the Intl. Conf. on Parallel Processing '88, I - Architecture, pp. 1-6, 1988

[4] Coffman E.G. Jr. (Ed.), *Computer and Job-Shop Scheduling Theory,* J. Wiley and Sons, New York, 1976

[5] Devis Yves, *Process allocation in a distributed computer system using a neural model,* MS. Thesis, State Univ. of Ghent, Report LEM-T9021, 1990

[6] Graham R.L., *Bounds on Certain Multiprocessing Anomalies,* SIAM Journal on Applied Mathematics, Vol. 17, 2, pp. 416-429, 1969

[7] Hu T.C., *Parallel sequencing and assembly line problems,* Op. Res., 9, 6, 841-848, 1961

[8] Hwang K., Xu Jian, *Mapping Partitioned program Modules onto Multicomputer Nodes Using Simulated Annealing,* Proceedings of the Intl. Conf. on Parallel Processing '90, II - Software, August 13-17, pp. 292-293, 1990

[9] Kaufman M.T., *An almost optimal algorithm for the assembly line problem,* IEEE Trans. on Computers-23, 11, 1169-1174, 1974

[10] Kirkpatrick S., Gelatt C.D., Vecchi M.P., *Optimization by Simulated Annealing,* Science, Vol. 220, Number 4598, May, pp. 671-680, 1983

[11] Lee S-Y., Aggarwal J.K., *A Mapping Strategy for Parallel Processing,* IEEE Trans. on Computers, Vol. 36, 4, pp. 433442, 1987

Table 1: Principal program characteristics. The C/C ratio represents the communication vs. computation ratio. Times are in μs

Program	Tasks	Average Duration	Average Commun.	C/C Ratio	Max. Speedup
Newton-Euler	95	9.12	3.96	43.0 %	7.86
Gauss-Jordan	111	84.77	6.85	8.1 %	9.14
FFT	73	72.74	6.41	8.8 %	40.85
Matrix Multiply	111	73.96	7.21	9.7 %	82.10

Table 2: Speedup figures for the benchmark programs. $(S_p)_{SA}$ and $(S_p)_{HLF}$ denote the speedup obtained with Simulated Anealing and with the HLF heuristic respectively.

Newton-Euler	w/o Comm.			with Comm.		
	$(S_p)_{SA}$	$(S_p)_{HLF}$	% gain	$(S_p)_{SA}$	$(S_p)_{HLF}$	% gain
Hypercube (8p)	7.20	6.90	4.4	5.6	4.9	14.3
Bus (8p)	7.20	6.90	4.4	6.2	5.2	11.5
Ring (9p)	8.00	8.00	0.0	5.5	3.6	52.8

Gauss-Jordan	w/o Comm.			with Comm.		
	$(S_p)_{SA}$	$(S_p)_{HLF}$	% gain	$(S_p)_{SA}$	$(S_p)_{HLF}$	% gain
Hypercube (8p)	6.67	6.67	0.0	4.80	4.64	3.5
Bus (8p)	6.76	6.67	1.4	4.93	4.74	3.9
Ring (9p)	8.25	8.25	0.0	5.02	4.77	5.0

Matrix Multiply	w/o Comm.			with Comm.		
	$(S_p)_{SA}$	$(S_p)_{HLF}$	% gain	$(S_p)_{SA}$	$(S_p)_{HLF}$	% gain
Hypercube (8p)	7.75	7.75	0.0	6.11	5.19	17.7
Bus (8p)	7.75	7.75	0.0	6.34	5.71	11.0
Ring (9p)	8.38	8.38	0.0	6.04	4.96	21.8

FFT	w/o Comm.			with Comm.		
	$(S_p)_{SA}$	$(S_p)_{HLF}$	% gain	$(S_p)_{SA}$	$(S_p)_{HLF}$	% gain
Hypercube (8p)	7.38	7.38	0.0	6.23	4.93	26.3
Bus (8p)	7.48	7.38	1.4	6.27	5.58	12.3
Ring (9p)	8.43	8.43	0.0	5.97	5.10	17.0

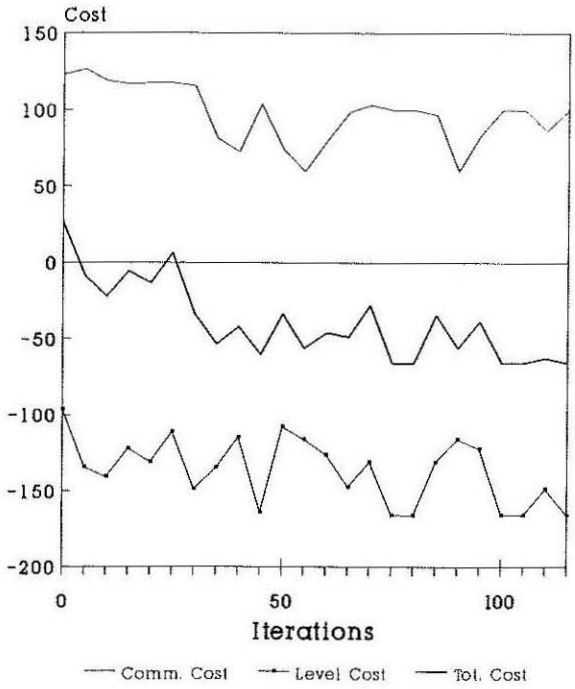

Figure 1: Cost trajectories F_b (level), F_c (communication) and F_{tot} (weighted sum) of a Newton-Euler annealing packet for an 8 node hypercube. The weights are $w_b = w_c = .5$

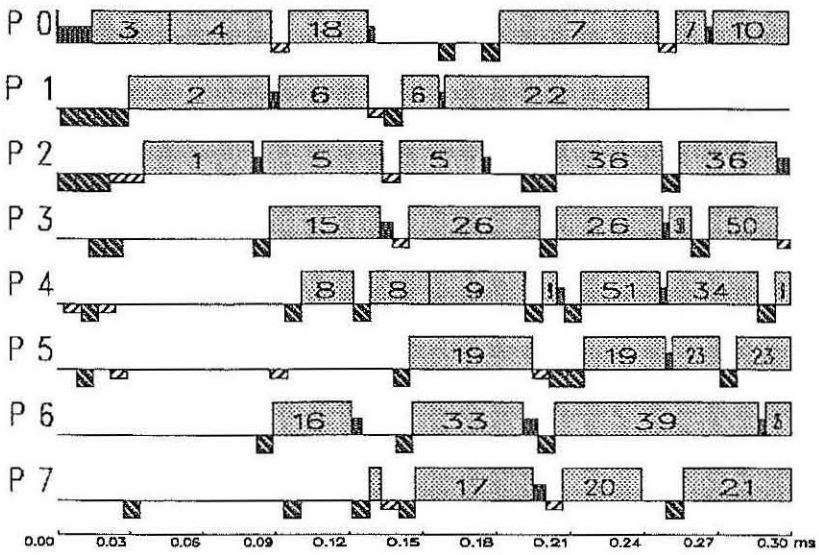

Figure 2: Gantt-chart of the Newton-Euler program on an 8 processor Hypercube (detail). Numbered blocks represent tasks, half-height blocks above and below the base line denote sending and receiving messages respectively, quarter-height blocks represent routing messages.